SCOPE GRAPH–BASED TYPE CHECKING FOR A SCALA SUBSET

RQ: Can we use a Haskell library for phased scope graph construction to type check a targeted Scala subset?

Radu Mihalachiuta
r.g.mihalachiuta@student.tudelft.nl

Responsible Professor: Casper Bach Poulsen

Supervisors: Aron Zwaan & Thomas Durieux

1 INTRODUCTION

Type checking ensures data type consistency, early error detection, while facing the inherent challenge of name binding.

Scope graphs offer a promising approach to handle intricate name resolution rules in type checkers, providing a uniform and language-independent model.

ADDRESSED CHALLENGES:

- Monotonicity ensures scope graph query stability by prohibiting the addition of critical edges, which are outgoing edges with the same label as a previously queried path [1].
- We address the challenge of precedence order on different resolution paths in Scala, specifically concerning objects and imports.

REFERENCES:
[1] ROUVOET, A., VAN ANTWERPEN, H., BACH POULSEN, C., KREBBERS, R., & VISSER, E. (2020). KNOWING WHEN TO ASK: SOUND SCHEDULING OF NAME RESOLUTION IN TYPE CHECKERS DERIVED FROM DECLARATIVE SPECIFICATIONS. PROCEEDINGS OF THE ACM ON PROGRAMMING LANGUAGES, 4(OOPSLA), 1-28.
[2] HENDRIK VAN ANTWERPEN, CASPER BACH POULSEN, ARJEN ROUVOET, AND EELCO VISSER. SCOPES AS TYPES. PROC. ACM PROGRAM. LANG., 2(OOPSLA), OCT 2018.

2 OBJECTIVES

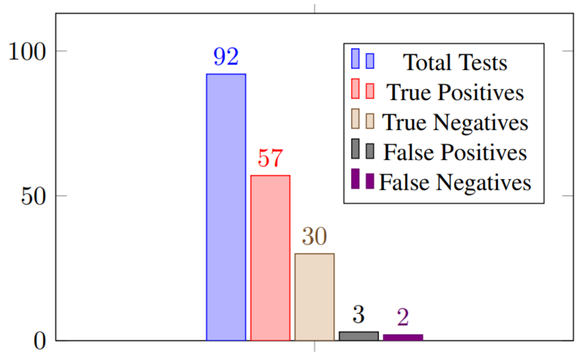
i) Develop a type checker that addresses precedence, name resolution complexities, and the challenge of monotonicity. We aim to combine automated scheduling convenience with the flexibility of the Haskell library.

ii) Explore the number of phases required for the type checker and investigate the impact of explicit phasing on our implementation.

iii) Qualitatively analyze and compare the declarativity and feature extensibility of our type checker against the mini-Statix approach by Rouvoet et al. [1].

4 RESULTS

- high level of reliability and precision;
- lack of support for sequenced imports and nested wildcard shadowing;



Experimental results for the extended mini-Statix test suite.

5 CONCLUSION

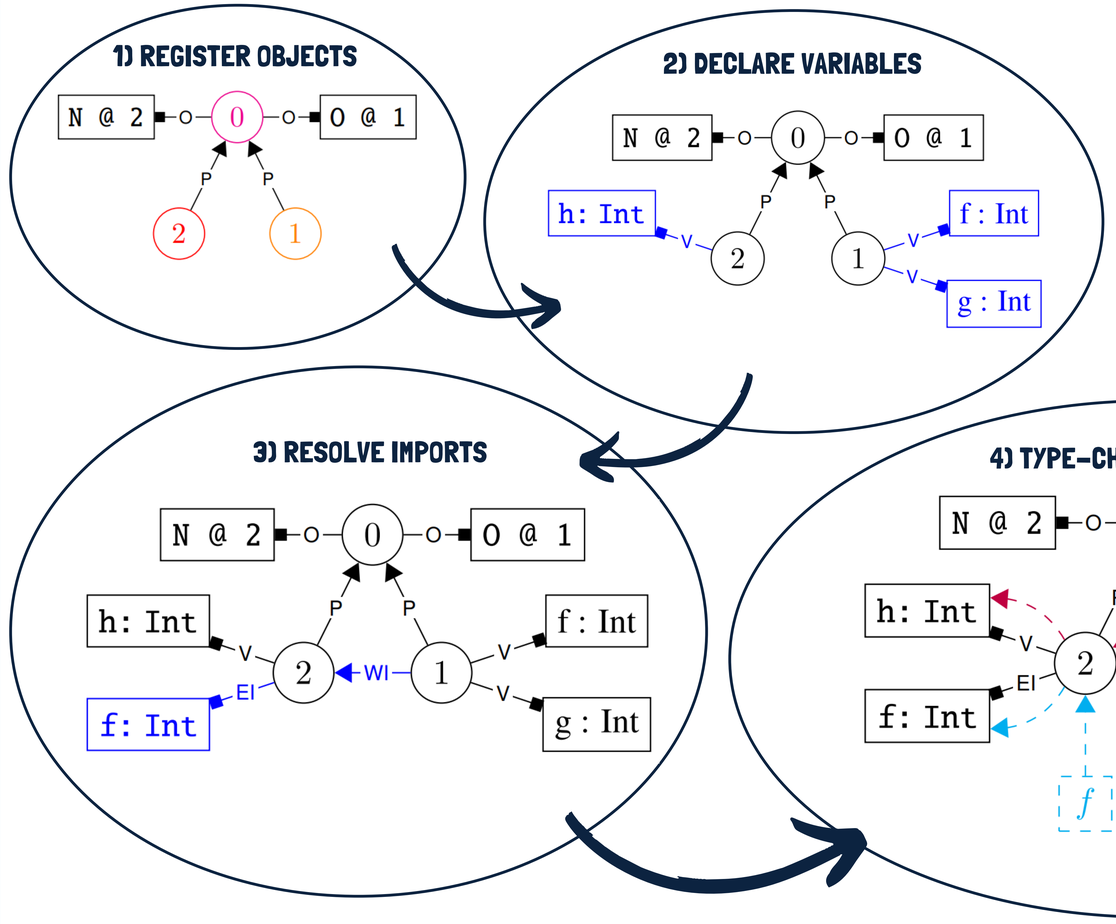
The promising, quantitative results are comparable to others from literature [1], [2]. Our approach is less declarative than mini-Statix, yet compensates through extensibility and modularity.

FUTURE WORK:

1) Addressing the unsupported test cases to achieve full coverage, improving accuracy and the usability of the type checker.
2) Incorporating additional features into the targeted Scala subset, thereby enhancing its extensibility.
3) Implementing similar type checkers for other languages.

3 PHASED TYPE CHECKER

The type checker is powered by a core algorithm consisting of four phases, which are exemplified through the program on the right.




```
object O {
  def f: Int = g
  import N._
  def g: Int = h
}

object N {
  import O.f
  def h: Int = f
}
```

Code example of mutually recursive definitions with imports, in circular dependencies.